Parallel Processing Letters

THE IMPACT OF MESSAGE-BUFFER ALIGNMENT ON COMMUNICATION PERFORMANCE

LEON ARBER
Performance and Architecture Lab
Computer and Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545, USA

and

SCOTT PAKIN
Performance and Architecture Lab
Computer and Computational Sciences Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545, USA

ABSTRACT

Of the many factors that contribute to communication performance, perhaps one of the least investigated is that of message-buffer alignment. Although the generally accepted practice is to page-align buffer memory for best performance, our studies show that the actual relationship of buffer alignment to communication performance cannot be expressed with such a simple formula. This paper presents a case study in which porting a simple network performance test from one language to another resulted in a large performance discrepancy even though both versions of the code consist primarily of calls to messaging-layer functions. Careful analysis of the two code versions revealed that the discrepancy relates to the alignment in memory of the message buffers. Further investigation revealed some surprising results about the impact of message-buffer alignment on communication performance: (1) different networks and node architectures prefer different buffer alignments; (2) page-aligned memory does not always give the best possible performance, and, in some cases, actually yields the *worst* possible performance; and, (3) on some systems, the most significant factor affecting network performance is the *relative* alignment of send and receive buffers with respect to each other.

Keywords: Data alignment, network performance, coNCePTuaL

1. Introduction

Network and messaging-layer performance tests are used for a variety of purposes, such as explaining or predicting system and application performance, monitoring improvements made during system deployment or messaging-layer development, and evaluating systems for purchase. Although there exist numerous ready-made communication microbenchmarks (e.g., the Pallas MPI Benchmarks [1], SKaMPI [2],

NetPIPE [3], and Mpptest [4]), there still exists a need for special-purpose, customized network performance tests. Special-purpose performance tests serve two important purposes: they help characterize and diagnose performance anomalies reported by more general-purpose tests; and, they can reproduce application-specific communication patterns, thereby providing insight into application performance.

coNCePTuaL [5] is a tool designed to simplify the acquisition and presentation of network performance data. The added benefits in simplicity, readability, and portability provided by coNCePTuaL made it worthwhile for the authors to port to coNCePTuaL a suite of existing performance tests used internally by the Performance and Architecture Lab (PAL) at Los Alamos National Laboratory. It was crucial that the coNCePTuaL versions exactly reproduce PAL's original C + MPI tests in order to fairly compare new measurements against historical data.

In fact, however, we observed a performance discrepancy between the two versions. Section 2 describes the problem and the initial steps taken to identify its cause. Section 3 explains how we exploited coNCePTuaL's ability to rapidly produce "disposable" special-purpose performance tests in order to drill down to the source of the problem and more precisely characterize various clusters' sensitivity to it. We relate our results to those found by others in Section 4. In Section 5 we present some avenues for follow-on research. Finally, we draw some conclusions from our discoveries in Section 6.

2. Simplifying Performance Testing

The Performance and Architecture Lab has been using an in-house suite of C + MPI network performance tests for many years. Because these tests have been in existence for a long time and have been run on a multitude of systems their results are well understood and are used as a historical reference against which future results can be compared. Although the test suite adequately performs the job it was meant to do, a coNCePTuaL [5] port would have a number of advantages including (1) the automatic production of "laboratory notebook" log files that include not just measurement data but also detailed information about the experimental setup [6]; (2) portability across messaging layers; and, (3) an enhanced ability to describe performance tests by treating coNCePTuaL programs, with their English-like syntax, as "executable pseudocode".

One of the first such tests we ported from C to coNCePTuaL was a basic ping-pong bandwidth test that measures bandwidth by repeatedly sending a message back-and-forth between two nodes and dividing the total number of bytes transmitted by the total time and reporting the result in bytes per microsecond (B/µs). The coNCePTuaL port aimed to replicate exactly the original C benchmark. The coNCePTuaL rewrite of the C benchmark is shown in its entirety in Listing 1.

The following is a brief description of the code. The first four statements of Listing 1 define command-line parameters and their default values. The main routine consists of a **let** statement (line 9) which binds the value of several variables for use

Listing 1. A coNCePTuaL ping-pong bandwidth test.

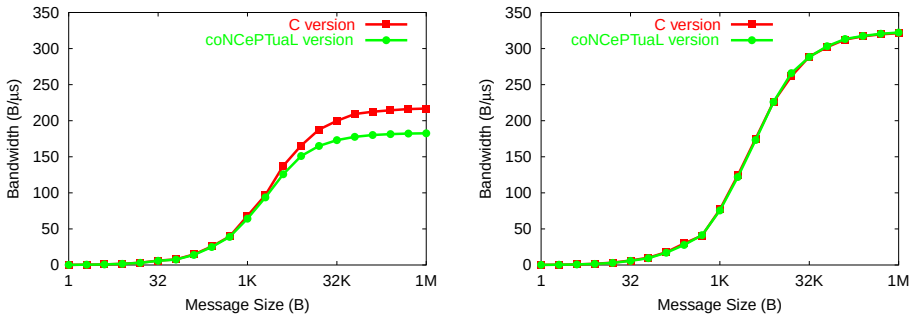
```
1  reps is "Number of repetitions" and comes from "--repetitions" or "-r"
       with default 10000.
2
3  minbytes is "Smallest number of bytes to send" and comes from
       "--minbytes" or "-m" with default 0.
4
5  maxbytes is "Largest number of bytes to send" and comes from
       "–maxbytes" or "-x" with default 512K.
6
7  inc is "Number of bytes to increment size of packet by after each
       set of iterations (0 means geometric progression)" and comes
       from "--inc" or "-i" with default 0.
8
9  Let multiplier be 1 if inc<>0 otherwise 2 and
10     seqstart be 1 if (minbytes=0 /\ inc=0) otherwise (minbytes*
           multiplier + inc) while
11   for each msgsize in {minbytes}, {seqstart, seqstart*multiplier +
         inc, (seqstart*multiplier + inc)*multiplier + inc, ...,
         maxbytes}
12   {
13     all tasks synchronize then
14     task 0 resets its counters then
15     for reps repetitions
16     {
17       task 0 sends a msgsize byte message to task 1 then
18       task 1 sends a msgsize byte message to task 0
19     }
20     then task 0 logs msgsize as "Message Size (bytes)" and
           elapsed_usecs/(2*reps) as "1/2 RTT (us)" and (reps*msgsize)
           /(elapsed_usecs/2) as "BW (B/us)"
21   }
```

in the subsequent loop nest. The outer **for** loop (line 11) determines the message sizes that the inner loop will use in its communications. The loop variable, *msgsize*, first takes on the value of the singleton {*minbytes*} followed by the numbers from *seqstart* to *maxbytes* in either an arithmetic or geometric progression (based on whether *inc* is 0). *minbytes* is handled separately in case the user sets it to 0 and specifies a geometric progression. With the default parameters, *msgsize* takes on the values $0, 1, 2, 4, \ldots, 524288$. Line 13 performs a global barrier operation. Line 14 makes task 0 reset its elapsed-time counter, byte counter, message counter, and other such counters. The program's inner loop (line 15) performs the actual ping-pong operations. The last statement in the program (line 20) tells coNCePTuaL to log several things to a file, including the size of the message in bytes, the average time in microseconds it took to send it (i.e., half of the round-trip time), and the bandwidth achieved in terms of bytes per microsecond (B/µs). See the coNCePTuaL User's Guide [7] for a more thorough explanation of the coNCePTuaL grammar.

As the ping-pong bandwidth test was simply a port of an existing program, the performance of the C and coNCePTuaL versions should have been identical.

Table 1. List of clusters tested and their specifications.

Cluster name	CPU (count, speed, model)	I/O bus (model, bits, MHz)	Network
IA-64/Elan 3 (also IA-64/GigE)	2 × 1.3 GHz Itanium 2	PCI 64/66	QsNet GigE
IA-32/Elan 3	2 × 1.13 GHz Pentium III	PCI 32/33	QsNet
x86-64/Elan 4	2 × 2 GHz Opteron	PCI-X 64/133	QsNet^{II}
IA-32/IBA	2 × 2.2 GHz Xeon	PCI-X 64/100	IBA 4X
IA-32/Myri	2 × 1 GHz Pentium III	PCI 64/33	Myrinet



(a) Benchmark results on the IA-64/ Elan 3 cluster

(b) Benchmark results on the IA-32/ Elan 3 cluster

Fig. 1. Two runs of a ping-pong bandwidth test

To verify that this was indeed the case, the coNCePTuaL code and the original C benchmark were run on the same set of machines, part of an Itanium 2 cluster interconnected with a Quadrics QsNet (Elan 3/Elite 3) network [8]. Table 1 presents the key characteristics of all of the clusters used in the preparation of this paper; the current discussion refers to the "IA-64/Elan 3" cluster listed in that table. The results of the run are shown in Figure 1(a). As is evident from the graph, although the results match to around 1,024 bytes they diverge from that point on, ultimately resulting in a performance discrepancy of 34 B/µs or 16%. Interestingly, on a cluster with a different CPU and node architecture but the same network (Table 1's IA-32/ Elan 3 cluster), the C and coNCePTuaL versions of the performance test yield qualitatively identical measurements (Figure 1(b)).

To track down the source of the performance discrepancy between the C and coNCePTuaL versions of the test, we migrated chunks of code from the hand-coded test into the coNCePTuaL-generated C code until the two versions of the performance test yielded the same performance. The linchpin turned out to be the rather innocuous memory-allocation code shown in Listing 2. When the coNCeP-

Listing 2. Memory allocation in the C version of the ping-pong performance test

```
1  if ((rbuf = (char *)malloc(maxNob ? maxNob : 8)) == NULL)
2  {
3      perror ("Failed memory allocation");
4      exit (1);
5  }
6
7  if ((tbuf = (char *)malloc(maxNob ? maxNob : 8)) == NULL)
8  {
9      perror ("Failed memory allocation");
10     exit (1);
11 }
```

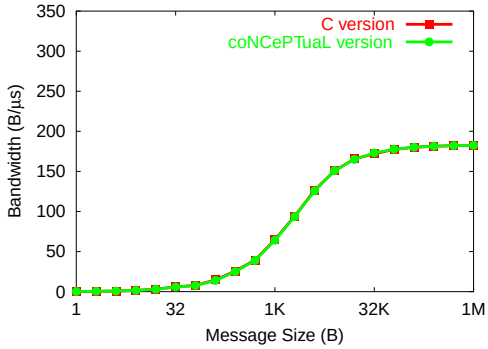


Fig. 2. Benchmark results with page-aligned memory on the IA-64/Elan 3 cluster

TUAL version of the test used the exact code shown in Listing 2 its performance matched the C version's. When it used its own—functionally equivalent—code its performance dropped by 34 B/µs. In addition, swapping the order of the two **if** statements in Listing 2 causes the hand-coded C version to lose 34 B/µs. Clearly, the performance test is sensitive to some subtle aspect of how memory is being allocated.

We hypothesized that the alignment in memory of the message buffers might be different between the code versions and that that might be the source of the performance discrepancy. To test this hypothesis, we rewrote the performance tests to page-align their message buffers by allocating them with `valloc()` instead of `malloc()`. When all communication buffers are page-aligned both the C and CONCEPTUAL versions observe the same performance on the IA-64/Elan 3 cluster, as shown in Figure 2. But there's a catch: both observe the same *poor* performance. Contrast Figure 2 with Figure 1(b)—data taken from a cluster with slower nodes. Because performance is different when running with page-aligned buffers (Figure 2) and without them (Figure 1(a)) we can conclude that buffer alignment does play a role in performance. However, it is striking that page-alignment yields suboptimal performance. In the following section we further investigate this phenomenon.

3. Memory Alignment

Although the results of the C and coNCePTuaL versions of the ping-pong performance test now matched on both platforms, page-aligning the memory of the C benchmark actually *decreased* its performance. This surprising outcome led us to question whether the effect is specific to the IA-64/Elan 3 cluster or if it is a more widespread problem. In addition, we wanted to gain a deeper understanding of the relationship between message-buffer alignment and communication performance.

3.1. Analysis of buffer alignment's effect on performance

coNCePTuaL is an ideal vehicle for carrying out such a study because it was designed to facilitate the rapid production of special-purpose performance tests. In fact, the language provides two keywords that are particularly apropos to alignment testing:

aligned Force the virtual address of a message buffer to be aligned at a multiple of a given number. For example, a "100 **byte aligned message**" can lie at addresses 0, 100, 200, 300, etc.

misaligned Force the virtual address of a message buffer to be misaligned from a page boundary by a given offset. On IA-32 and x86-64 systems a page is 4 KB; on IA-64 systems a page is 16 KB. For example, a "100 **byte misaligned message**" on IA-32/Elan 3 can lie at addresses 100, 4196, 8292, 12388, etc.

The test we devised for measuring a network's sensitivity to buffer alignment is presented in its entirety in Listings 3–5. The first part of the test, Listing 3, measures ping-pong bandwidth when the send and receive buffers are aligned equally, meaning that each is **aligned** to a multiple of the *alignment* variable which ranges in 4-byte increments from 0 to *maxalign*. The second and third parts of the test—still within the loop over *alignment*—respectively page-align the receive buffers while varying how much the send buffers are **misaligned** (Listing 4) and page-align the send buffers while varying how much the receive buffers are **misaligned** (Listing 5). Between tests, the program touches (i.e., reads and writes) every word of a large memory region in order to flush the system's various memory caches.

One aspect of coNCePTuaL that may need some explanation is that **send** statements implicitly post a matching **receive** with matching parameters. The **unsuspecting** keyword suppresses the implicit **receive**. Listings 4–5 use **unsuspecting** to specify different alignment parameters for the sender and receiver.[a]

All of the graphs of alignment results in this section are presented in pairs. The first graph in each pair starts the y axis at 0 B/µs so as not to exaggerate the impact of varying buffer alignments. The second graph in each pair provides a close-up look at the range of values actually observed in order to emphasize details

[a] A relatively recent addition to coNCePTuaL achieves the same effect using more natural semantics, as will be seen in Listing 6.

Listing 3. coNCePTuaL memory-alignment test (Part 1 of 3: equal alignment)

```
reps is "Number of repetitions" and comes from "--repetitions" or "-r"
    with default 1000.

maxalign is "Maximum alignment/misalignment to test with" and comes
     from "--maxalign" or "-m" with default 256.

msgsize is "Size of message to test with" and comes from "--size" or
    "-s" with default 512K.

For each alignment in {0, 4, 8, 12, ..., maxalign}
{
  all tasks synchronize then
  task 0 resets its counters then

  # Test the effects of varying equal alignment.
  for reps repetitions
  {
    task 0 sends a msgsize byte alignment byte aligned message to
        task 1 then
    task 1 sends a msgsize byte alignment byte aligned message to
        task 0
  }
  then task 0 logs msgsize as "Message size (bytes)" and alignment
      as "Alignment (bytes)" and (msgsize*reps*2)/elapsed_usecs as
      "aligned BW (B/us)" then
```

that are not apparent from the long-range view. To improve readability, all x axes range from 0 to 256 bytes, not to the full page size; when relevant, the text describes the graph beyond the 256-byte limit. Each graph contains three curves: *Equally aligned*, in which both the send buffer and the receive buffer lie at a virtual address that is a multiple of the value on the x axis; *Send buffer misaligned*, in which the receive buffer is page-aligned and the send buffer lies at a virtual address x bytes past a page boundary; and, *Receive buffer misaligned*, in which the send buffer is page-aligned and the receive buffer lies at a virtual address x bytes past a page boundary. Note that both buffers being page-aligned is represented by $x = 0$ on the *Send buffer misaligned* and *Receive buffer misaligned* curves. All experiments used the network vendor's MPI implementation, generally an MPICH [9] derivative.

The first set of results presented are for the IA-64/Elan 3 and IA-32/Elan 3 clusters (Figures 3 and 4). Figure 3 confirms the observations made in Section 2: performance on the IA-64/Elan 3 cluster varies by 36 B/µs (180–216 B/µs) based on the alignment of the message buffers; and, page-aligned buffers yield bandwidths near the low end of observed values. On the IA-32/Elan 3 cluster, varying buffer alignment exhibits relatively little impact, only up to a maximum performance loss of only 2.3% of the peak bandwdith. However, as both Figures 3(b) and 4(b) clearly show, the performance-degradation pattern of the misaligned buffers recurs every 64 bytes and not in multiples of the page size as one might suspect. This 64-byte performance cycle is apparently a characteristic of the QsNet network. Communica-

Listing 4. coNCePTuaL memory-alignment test (Part 2 of 3: misaligned sender)

```
19  # Test the effects of keeping the receive buffer page aligned
        while varying the misalignment of the send buffer.
20  all tasks touch a 10 megabyte memory region then
21  all tasks synchronize then
22  task 0 resets its counters then
23  for reps repetitions
24  {
25    task 1 receives a msgsize byte page aligned message from task 0
          then
26    task 0 sends a msgsize byte alignment byte misaligned message
          to unsuspecting task 1 then
27    task 0 receives a msgsize byte page aligned message from task 1
          then
28    task 1 sends a msgsize byte alignment byte misaligned message
          to unsuspecting task 0
29  }
30  then task 0 logs (msgsize*reps*2)/elapsed_usecs as "misaligned BW
        (B/us)" then
```

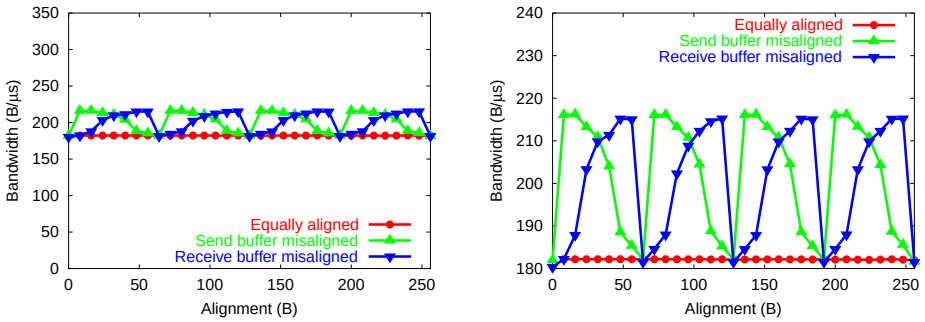
Listing 5. coNCePTuaL memory-alignment test (Part 3 of 3: misaligned receiver)

```
31  # Test the effects of keeping the send buffer page-aligned while
        varying the misalignment of the receive buffer.
32  all tasks touch a 10 megabyte memory region then
33  all tasks synchronize then
34  task 0 resets its counters then
35  for reps repetitions
36  {
37    task 1 receives a msgsize byte alignment byte misaligned
          message from task 0 then
38    task 0 sends a msgsize byte page aligned message to
          unsuspecting task 1 then
39    task 0 receives a msgsize byte alignment byte misaligned
          message from task 1 then
40    task 1 sends a msgsize byte page aligned message to
          unsuspecting task 0
41  }
42  then task 0 logs (msgsize*reps*2)/elapsed_usecs as "misaligned BW
        (sender page-aligned) (B/us)"
43 }
```

tion in QsNet is based on a hardware RDMA "put" primitive. An implementation artifact of QsNet's data-prefetching algorithm—which is necessarily cognizant of the virtual memory addresses on both ends of the transmission—is causing the observed sensitivity of performance to buffer alignment.

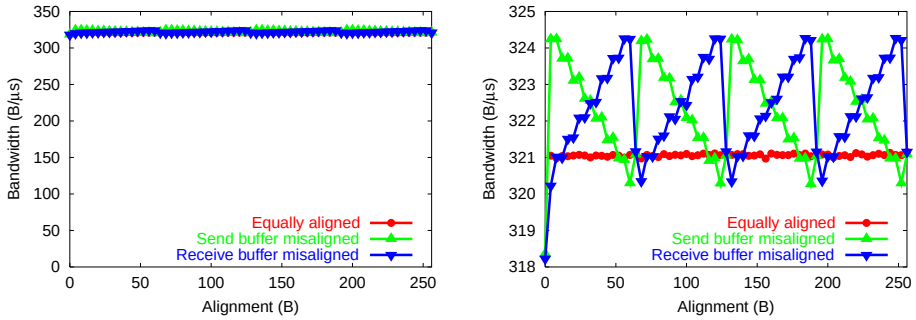
3.2. The significance of relative alignment

One important observation to make for Figures 3 and 4 is that when the send and receive buffers are aligned equally in memory (i.e., each to a multiple of the number shown on the x axis) performance is consistently poor. This observation



(a) y axis starts at 0 B/µs (b) y axis starts at 180 B/µs

Fig. 3. Alignment results for the IA-64/Elan 3 cluster



(a) y axis starts at 0 B/µs (b) y axis starts at 318 B/µs

Fig. 4. Alignment results for the IA-32/Elan 3 cluster

leads one to wonder if good performance depends upon the relative alignment of the send and receive buffers to each other. Figures 3–4 demonstrate that good performance can be achieved when one buffer is page-aligned and the other is offset from a page boundary but is the impact on performance in fact a function of the *difference* in alignment between the two buffers?

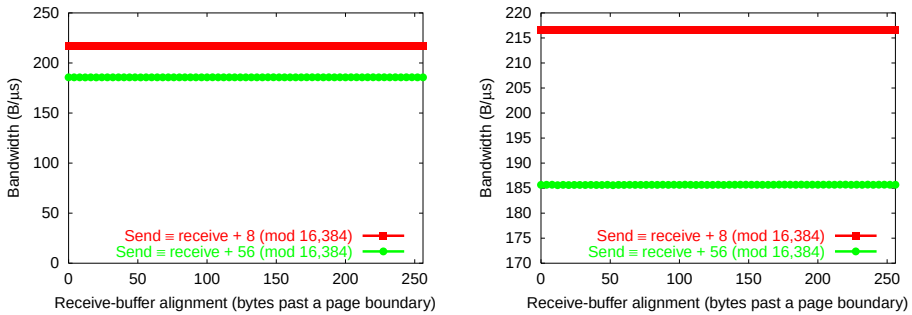
To answer that question, we devised another special-purpose performance test using coNCePTuaL. Listing 6 presents the complete coNCePTuaL source code for a ping-pong bandwidth test that aligns the receive buffer *alignment* bytes past a page boundary and the send buffer {*alignment* + *misalignment*} bytes past a page boundary. We ran this new performance test on the IA-64/Elan 3 cluster using *misalignment* values of 8 bytes and 56 bytes. The numbers 8 and 56 were selected because Figure 3 displays a peak every time the send buffer is 8 bytes (modulo 64) ahead of the page-aligned receive buffer and a valley every time the send buffer

Listing 6. coNCePTuaL test of relative buffer alignment

```
reps is "Number of repetitions" and comes from "--repetitions" or "
    -r" with default 1000.

maxalign is "Maximum starting alignment to test with in bytes" and
    comes from "--maxalign" or "-m" with default 16K.

msgsize is "Size of message to test with" and comes from "--size"
    or "-s" with default 1M.

misalignment is "Relative misalignment to test with in bytes" and
    comes from "--misalign" or "-g" with default 8.

For each alignment in {0, 4, 8, ..., maxalign}
{
  all tasks synchronize then
  task 0 resets its counters then
  for reps repetitions
  {
    task 0 sends a msgsize byte alignment+misalignment byte
        misaligned message to task 1 who receives it as an
        alignment byte misaligned message then
    task 1 sends a msgsize byte alignment+misalignment byte
        misaligned message to task 0 who receives it as an
        alignment byte misaligned message
  }
  then task 0 logs alignment as "Alignment (B)" and (msgsize*reps
      *2)/elapsed_usecs as "Bandwidth (B/us)"
}
```

is 56 bytes (modulo 64) ahead of the page-aligned receive buffer. Figure 5 clearly shows that performance on the IA-64/Elan 3 cluster is sensitive only to the *relative* distance between the send and receive buffers modulo the page size; the *absolute* alignment relative to a page boundary is inconsequential. Although Figure 5 plots the results only out to 256 bytes the straight-line pattern continues at least to 16,384 bytes, which is the IA-64 page size.

The data confirm that the Elan 3 network performs best when the send buffer lies a number of full pages *plus* 8 bytes ahead of the corresponding receive buffer and performs worst when the send buffer lies a number of full pages *minus* 8 bytes ahead of the corresponding receive buffer. To determine if this performance characteristic explains the performance discrepancy observed in Section 2 we ran the C and coNCePTuaL versions of the original ping-pong bandwidth test within a debugger and dumped the virtual memory addresses of the send and receive buffers used for transmitting 1 MB messages. Sure enough, on the IA-64/Elan 3 cluster the C version happened to allocate the receive buffer at virtual address $\texttt{6000000000BA3890}_{\text{hex}}$ and the send buffer at virtual address $\texttt{6000000000CA38A0}_{\text{hex}}$—a "good" difference of the send buffer being 1 MB + 16 B ahead of the receive buffer. (The extra 16 bytes represents two 64-bit integers' worth of `malloc()` housekeeping information.) The coNCePTuaL version, which by default shares buffers across non-concurrent op-



(a) y axis starts at 0 B/µs (b) y axis starts at 170 B/µs

Fig. 5. Variable receive-buffer address, fixed relative offset of send buffer (IA-64/Elan 3 cluster)

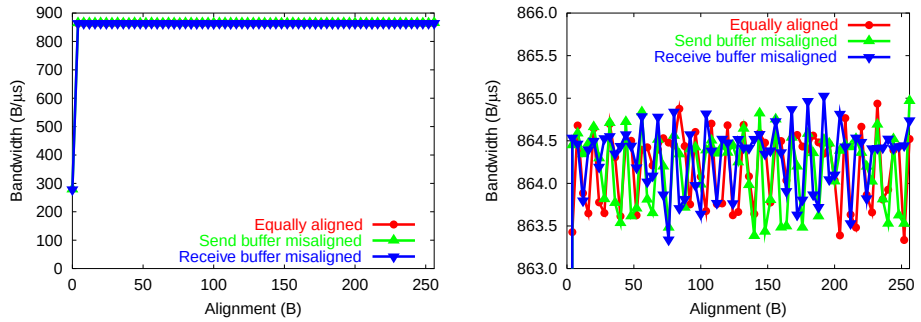
erations, allocated both buffers at virtual address `6000000000BBC6F0`$_{\text{hex}}$. This represents a "bad" difference of 0 B. We can therefore conclude that the performance discrepancy presented in Section 2 was, in fact, caused by the relative alignment of the send and receive buffers. Recall that we also observed in Section 2 that swapping the order of the `malloc()` calls in Listing 2 degraded performance. As we now know, allocating the send buffer before allocating the receive buffer places the receiver at a "bad" offset of 1 MB + 16 B ahead of the sender.

3.3. Other Networks

Having quantified the alignments that result in good and bad performance on QsNet we chose to expand our study to other high-speed interconnects. The following paragraphs present and explain the results we took on Quadrics's QsNet$^{\text{II}}$ (Elan 4/Elite 4) [10], Mellanox's InfiniBand 4X [11] implementation, Myricom's Myrinet-2000 [12], and Broadcom's Gigabit Ethernet [13] implementation (NetXtreme BCM5701 cards).

QsNet$^{\text{II}}$ Figure 6 shows that the x86-64/Elan 4 cluster (described in Table 1) performs poorly when the send and receive buffers are page-aligned but performs well for any other alignment and with negligible variation. However, further investigation revealed that the initial performance drop is an artifact not related to buffer alignment. On the QsNet$^{\text{II}}$, a message that is allocated but not initialized exacts a severe performance penalty (∼2.5 s for a 1 MB message) when first accessed because of the way the network interface demand-pages virtual memory from the host. Re-running the x86-64/Elan 4 experiment in the reverse order, from largest to smallest alignment, penalized the first alignment tested and returned a bandwidth of 864 B/µs for the 0 B misalignment cases.

A possible explanation of why the IA-32/Elan 3 and x86-64/Elan 4 clusters are



(a) y axis starts at 0 B/µs

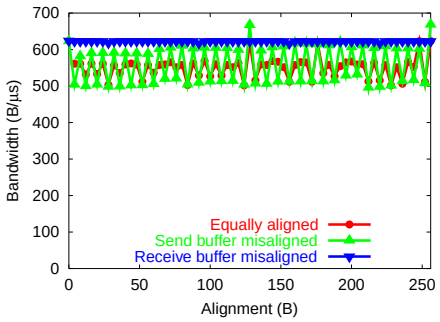
(b) y axis starts at 863 B/µs

Fig. 6. Alignment test results for the x86-64/Elan 4 cluster

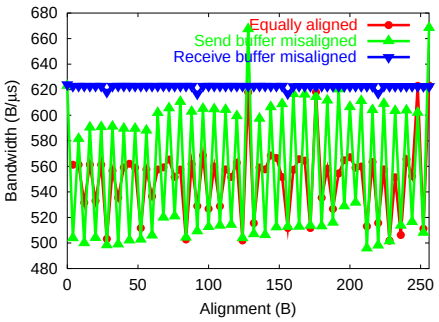
largely unaffected by buffer alignment but the IA-64/Elan 3 cluster is sensitive to it involves the matching between the I/O bus and network card. The IA-32/Elan 3 cluster runs a PCI network card on a PCI bus; the x86-64/Elan 4 cluster runs a PCI-X network card on a PCI-X bus; but, the IA-64/Elan 3 cluster runs a PCI network card on a PCI-X bus (specifically, the one in Hewlett-Packard's zx1 chipset). We believe that the PCI/PCI-X interface on the IA-64/Elan 3 cluster exaggerates the QsNet's sensitivity to buffer alignment although the exact reason why this occurs needs to be investigated further.

InfiniBand The IA-32/IBA cluster (described in Table 1), which is interconnected using Mellanox's InfiniBand 4X [11] implementation, is highly sensitive to message-buffer alignment. As Figure 7 shows, based on how the send and receive buffers are aligned, performance can drop over 25% from the peak measured performance of 668 B/µs to a low of 496 B/µs. However, the IA-32/IBA cluster favors different alignments from the also-sensitive IA-64/Elan 3 cluster. On the IA-32/IBA cluster, the receive buffer's misalignment is largely inconsequential; performance is determined primarily (but not completely; cf. the *Equally aligned* curve) by the send buffer's misalignment relative to a page boundary. Furthermore, performance depends heavily on which multiple of a 4-byte boundary the send buffer is aligned to. Even multiples (i.e., receive-buffer address + 8 bytes, + 16 bytes, + 24 bytes, etc.) result in almost 100 B/µs better performance than odd multiples (i.e., receive-buffer address + 4 bytes, + 12 bytes, + 20 bytes, etc.)—an average of 607 B/µs vs. an average of only 509 B/µs. Although the exact values appear erratic in Figure 7(b), by extending the x axis, Figure 7(c) shows that the *Send buffer misaligned* curve repeats a pattern every 1024 bytes.

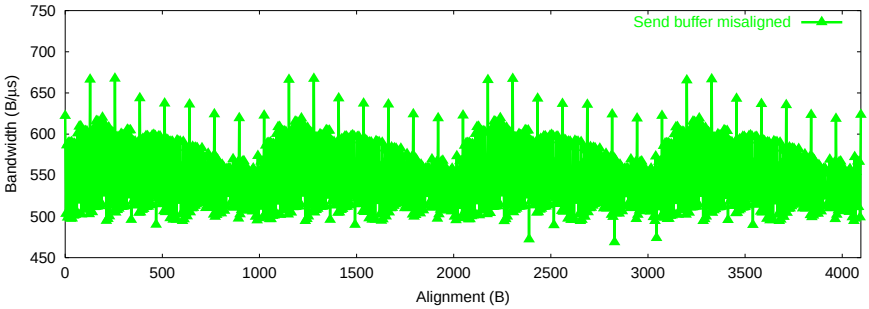
Myrinet The IA-32/Myri cluster, interconnected using Myricom's Myrinet [12] network exhibits essentially the same bandwidth regardless of buffers alignment



(a) y axis starts at 0 B/µs



(b) y axis starts at 480 B/µs

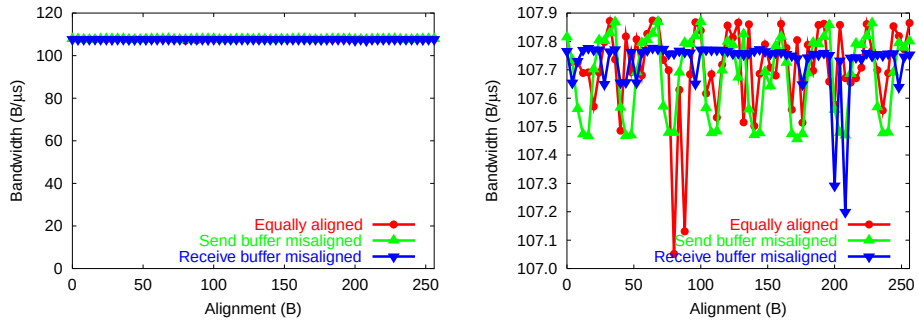


(c) x axis extended to 4000 B

Fig. 7. Alignment test results for the IA-32/IBA cluster

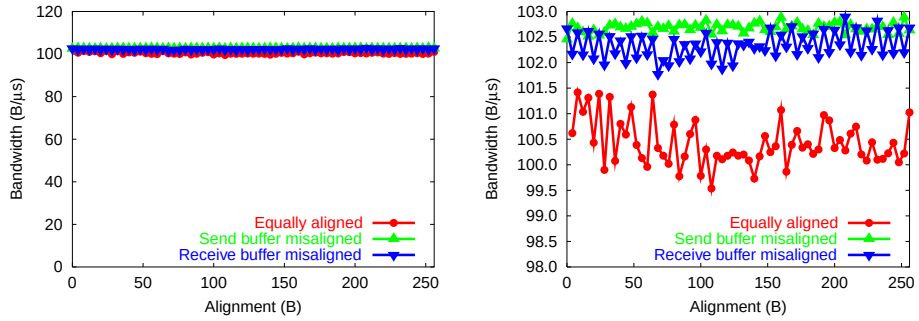
(Figure 8). Although Figure 8(b) indicates that regular alignment effects are exhibited when the receive buffer is page-aligned and the send buffer is relatively misaligned, these account for a performance loss of under 1% of the measured peak performance. The downward spikes in Figure 8(b) are probably caused by interference from other users—the IA-32/Myri cluster was running in only a partially dedicated mode—and are unlikely to be significant.

Gigabit Ethernet As a final comparison, the alignment test was also run over the Gigabit Ethernet network [13] in the IA-64/GigE cluster. (As indicated by Table 1, the IA-64/GigE cluster is the same cluster as the IA-64/Elan 3 cluster; it simply has both a QsNet network and a Gigabit Ethernet network.) What makes the IA-64/GigE configuration worth investigating is that MPI runs over the heavyweight TCP/IP protocol instead of bypassing the operating system as it does in all of the other configurations we tested. Figure 9 shows that the IA-64/GigE cluster observes little performance variance even though the QsNet$^{\text{II}}$ network in the same



(a) y axis starts at 0 B/µs (b) y axis starts at 107 B/µs

Fig. 8. Alignment test results for the IA-32/Myri cluster



(a) y axis starts at 0 B/µs (b) y axis starts at 98 B/µs

Fig. 9. Alignment test results for Gigabit Ethernet

cluster is highly sensitive to message-buffer alignment. The detail view presented by Figure 9(b) does indicate that when the send and receive buffers are aligned equally relative to a page boundary, performance is consistently worse than when the two are relatively misaligned. However, this difference accounts for a loss of less than 2% of the measured peak bandwidth.

Although the effect is slight, page-aligned receive buffers/misaligned send buffers do outperform page-aligned send buffers/misaligned receive buffers—the opposite of what was observed on the IA-32/IBA cluster. As with Figure 7, we extended the curves shown in Figure 9 out to 16,384 bytes but in this case did not observe any periodic behavior.

4. Related Work

While we are unaware of any prior work that specifically examines the effects of buffer alignment on network performance, the closest study to ours in spirit involves a series of tests performed with NetPIPE [3]. Like coNCePTuaL, NetPIPE is designed to test network performance; and, like coNCePTuaL, it is separated into a messaging-layer-independent part and a messaging-layer-dependent part. However, coNCePTuaL is designed to be a general-purpose testing framework while NetPIPE is specialized to a predefined set of tests. Using NetPIPE, Snell et al. [3] discovered a discrepancy in network performance caused by buffer alignment. When running TCP/IP over an ATM network, the bandwidth of unaligned messages (not explained in Snell et al.'s paper beyond implying "not page-aligned") is poor for only a particular range of message sizes while page-aligned messages perform well in that same range of sizes. In contrast, we observed that—on a different set of networks, admittedly—performance can be more sensitive to the *relative* alignment of the send and receive buffers on a node and that page-aligned buffers sometimes result in worst-case performance.

5. Future Work

Although this paper presents the effects of buffer alignment on various platforms, the precise causes of these effects are still unknown. Some possible sources include the network interface's architecture or implementation, memory and I/O bus interactions, and artifacts of the various messaging layers. An important follow-on study would be to isolate each of those components in turn in an attempt to identify the sources of communication performance's sensitivity to buffer alignment.

A related investigation is to extend our study to more tightly coupled parallel computers. Such systems may include special-purpose communication hardware (e.g., bulk-transfer engines) which can be a source of alignment sensitivity.

A final avenue for future research would be to design and implement a tool which, given the output from an alignment test running on a particular cluster, would optimize an application's communication buffers to ensure optimal performance. Such a tool could become an integral part of the software optimization process, especially on platforms known to be highly susceptible to alignment effects.

6. Conclusions

It is widely known that the alignment of message buffers can affect communication performance. However, the data presented in this paper contradict the notion that page-aligned buffers yield optimal performance. In fact, we have shown that different networks and node architectures favor different buffer alignments. On some clusters, page-aligned buffers make a simple ping-pong performance test report its *worst* performance. In most cases, the best performance is achieved only when either the send buffer or the receive buffer is offset from a page boundary, although the amount differs from cluster to cluster. Table 2 summarizes the impact

of message-buffer alignment on ping-pong bandwidth.[b] As shown by that table, the impact can be significant—up to a 25.8% loss in performance on the IA-32/IBA cluster from the most to the least favorable buffer alignment.

It would be worthwhile for an alignment test such as the one proposed in this paper to be integrated into existing network performance tests such as the Pallas MPI Benchmarks [1] or SKaMPI [2]. Knowing ahead of time how much of a factor alignment can play in network performance would lead to better predictions of application performance and can prove useful in tracking down performance anomalies.

In short, the interplay of cluster architecture and message-buffer alignment is critical to understanding communication performance. The results presented in this paper indicate that expressing communication performance as a function of message-buffer alignment and cluster architecture is a requirement for deriving realistic expectations of a network's delivered performance.

The coNCePTuaL software package used throughout the course of this case study is freely available from `http://conceptual.sourceforge.net/`.

Table 2. Summary of performance as a function of buffer alignment

Cluster	Best perf.		Worst perf.		Δ%
	B/µs	Pattern	B/µs	Pattern	
IA-64/Elan 3	216.2	$S \equiv R + 16 \pmod{64}$	180.3	$S \equiv R \pmod{64}$	16.6
IA-32/Elan 3	324.3	$S \equiv R + 16 \pmod{64}$	317.0	$S \equiv R \pmod{64}$	2.3
x86-64/Elan 4	*865.0*	*No pattern*	*278.9*	*S, R page-aligned*	*67.8*
	865.0	No pattern	863.3	No pattern	0.2
IA-32/IBA	668.4	$S \equiv 256 \pmod{1K}$	495.8	$S \equiv 468 \pmod{1K}$	25.8
IA-32/Myri	107.9	$S \equiv 4 \pmod{32}$	107.1	$S \equiv 12 \pmod{32}$	0.7
IA-64/GigE	102.9	$R \equiv 0 \pmod{16K}$	99.5	$S \equiv R \pmod{16K} \wedge R \not\equiv 0 \pmod{16K}$	3.3

Acknowledgements

This work was supported primarily by the ASC program at Los Alamos National Laboratory and in part by the Defense Advanced Research Projects Agency under the High Productivity Computing Systems Program. Los Alamos National Laboratory is operated by the University of California for the U.S. Department of Energy under contract W-7405-ENG-36. The authors would like to thank David Addison for explaining the QsNet^{II} first-message performance penalty and for his early InfiniBand alignment studies; Fabrizio Petrini for his explanation of QsNet's sensitivity to relative buffer misalignment; the rest of the PAL team for their insights throughout the course of this study; Michael Heath and CSE for the use of their IA-32/Myri cluster; and, LANL's CCN for the use of their IA-32/IBA cluster.

[b] The italic IA-64/Elan 3 row represents the initial measurement; the roman IA-64/Elan 3 row is the result of re-running the test in reverse order as described on page 11.

[1] Pallas, GmbH. *Pallas MPI Benchmarks—PMB, Part MPI-1*, 2000. Available from `ftp://ftp.pallas.com/pub/PALLAS/PMB/PMB-MPI1.pdf`.

[2] R. Reussner, P. Sanders, L. Prechelt and M. Müller. SKaMPI: A detailed, accurate MPI benchmark, in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: Proc. 5th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'98)*, eds. V. Alexandrov and J. Dongarra (Springer-Verlag, Liverpool, United Kingdom, 1998), vol. 1497 of *Lecture Notes in Computer Science* 52–59. ISBN 3-540-65041-5. Available from `http://www.mpi-sb.mpg.de/~sanders/papers/europvm-mpi98.ps.gz`.

[3] Q. O. Snell, A. R. Mikler and J. L. Gustafson. NetPIPE: A network protocol independent performance evaluator, in *Proc. 1996 ISMM Int. Conf. on Intelligent Information Management Systems*, ed. J. S. Wong (ACTA Press, Washington, DC, 1996) ISBN 0-88986-207-9. Available from `http://www.scl.ameslab.gov/netpipe/paper/netpipe.ps`.

[4] W. Gropp and E. Lusk. Reproducible measurements of MPI performance characteristics, in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: Proc. 6th European PVM/MPI Users' Group Meeting (EuroPVM/MPI'99)*, eds. J. Dongarra, E. Luque and T. Margalef (Springer-Verlag, Barcelona, Spain, 1999), vol. 1697 of *Lecture Notes in Computer Science* 11–18. Available from `http://www.mcs.anl.gov/~gropp/bib/papers/1999/pvmmpi99/mpptest.pdf`.

[5] S. Pakin. coNCePTuaL: A network correctness and performance testing language, in *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, New Mexico, 2004, Available from `http://www.c3.lanl.gov/~pakin/papers/ipdps2004.pdf`.

[6] S. Pakin. Reproducible network benchmarks with coNCePTuaL, in *Proc. Euro-Par 2004*, Pisa, Italy, 2004, Available from `http://www.c3.lanl.gov/~pakin/papers/europar2004.pdf`.

[7] S. Pakin. coNCePTuaL user's guide. Technical Report LA-UR 03-7356, Los Alamos National Laboratory, Los Alamos, New Mexico, 2003. Available from `http://www.c3.lanl.gov/~pakin/software/conceptual/conceptual.pdf`.

[8] F. Petrini, W. Feng, A. Hoisie, S. Coll and E. Frachtenberg. The Quadrics network: High-performance clustering technology, *IEEE Micro* **22** (2002) 46–57. Available from `http://www.c3.lanl.gov/~fabrizio/papers/ieeemicro.pdf`.

[9] W. Gropp, E. Lusk, N. Doss and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing* **22** (1996) 789–828. Available from `ftp://ftp.mcs.anl.gov/pub/mpi/mpicharticle.ps`.

[10] J. Beecroft, D. Addison, F. Petrini and M. McLaren. Quadrics QsNet II: A network for supercomputing applications, in *Proc. Hot Chips 15*, Palo Alto, California, 2003, Available from `http://www.c3.lanl.gov/~fabrizio/papers/hot03.pdf`.

[11] Mellanox Technologies, Inc. Introduction to InfiniBand, 2003. Document Number 2003WP. Available from `http://www.mellanox.com/technology/shared/IB_Intro_WP_190.pdf`.

[12] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawick, C. L. Seitz, J. N. Seizovic and W.-K. Su. Myrinet: A gigabit-per-second local area network, *IEEE Micro* **15** (1995) 29–36. Available from `http://www.myri.com/research/publications/Hot.ps`.

[13] LAN/MAN Standards Committee. Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. IEEE Standard 802.3, IEEE Computer Society, Technical Committee on Computer Communications, New York, New York, 2002. ISBN 0-7381-3089-3. Available from `http://standards.ieee.org/getieee802/download/802.3-2002.pdf`.